

CLRS 15.4

= “recursão-com-tabela”
 = transformação inteligente de recursão em iteração

$\langle z_1, \dots, z_k \rangle$ é **subsequência** de $\langle x_1, \dots, x_m \rangle$
 se existem índices $i_1 < \dots < i_k$ tais que

$$z_1 = x_{i_1} \quad \dots \quad z_k = x_{i_k}$$

EXEMPLOS:

$\langle 5, 9, 2, 7 \rangle$ é subsequência de $\langle 9, 5, 6, 9, 6, 2, 7, 3 \rangle$

$\langle A, A, D, A, A \rangle$ é subsequência de $\langle A, B, R, A, C, A, D, A, B, R, A \rangle$

A	A	D	A	A
A	B	R	A	C
A	D	A	B	R
A	A	B	R	A

Exercício

Problema: Decidir se $Z[1..m]$ é subsequência de $X[1..n]$

SUB-SEQ (Z, m, X, n)

```

1  i ← m
2  j ← n
3  enquanto i ≥ 1 e j ≥ 1 faça
4      se Z[i] = X[j]
5          então i ← i - 1
6      j ← j - 1
7  se i ≥ 1
8      então devolva “não é subsequência”
9      senão devolva “é subsequência”
    
```

Consumo de tempo é $O(n)$ e $\Omega(\min\{m, n\})$.

Invariantes:

- (i0) $Z[i+1..m]$ é subsequência de $X[j+1..n]$
- (i1) $Z[i..m]$ **não** é subsequência de $X[j+1..n]$

Subsequência comum máxima

Z é **subseq comum** de X e Y

se Z é subsequência de X e de Y

ssco = subseq comum

Exemplos: $X = A B C B D A B$
 $Y = B D C A B A$
 ssco = $B C A$

Exemplos: $X = A B C B D A B$
 $Y = B D C A B A$
 ssco = $B C A$
 Outra ssco = $B D A B$

Problema

Problema: Encontrar uma **ssco máxima** de X e Y .

Exemplos: $X = A B C B D A B$

$Y = B D C A B A$

ssco = $B C A$

ssco **maximal** = $A B A$

ssco **máxima** = $B C A B$

Outra sscó máxima = $B D A B$

LCS = Longest Common Subsequence

diff

> more abracadabra

A

B

R

A

C

A

D

A

B

R

A

> more yabbadabbadoo

Y

A

B

B

A

D

A

B

B

A

D

D

0

diff -u abracadabra yabbadabbadoo

+Y

A

B

-R

-A

-C

+B

A

D

A

B

-R

+B

A

+D

+0

+0

Subestrutura ótima

Suponha que $Z[1..k]$ é **ssco máxima** de $X[1..m]$ e $Y[1..n]$.

- ▶ Se $X[m] = Y[n]$,
então $Z[k] = X[m] = Y[n]$ e
 $Z[1..k-1]$ é sscó máxima de $X[1..m-1]$ e $Y[1..n-1]$.
- ▶ Se $X[m] \neq Y[n]$,
então $Z[k] \neq X[m]$ implica que
 $Z[1..k]$ é sscó máxima de $X[1..m-1]$ e $Y[1..n]$.
- ▶ Se $X[m] \neq Y[n]$,
então $Z[k] \neq Y[n]$ implica que
 $Z[1..k]$ é sscó máxima de $X[1..m]$ e $Y[1..n-1]$.

Simplificação

Problema: encontrar o **comprimento** de uma sscó máxima.

$c[i, j]$ = comprimento de uma sscó máxima de $X[1..i]$ e $Y[1..j]$

Recorrência:

$$c[0, j] = c[i, 0] = 0$$

$$c[i, j] = c[i-1, j-1] + 1 \text{ se } X[i] = Y[j]$$

$$c[i, j] = \max(c[i, j-1], c[i-1, j]) \text{ se } X[i] \neq Y[j]$$

Algoritmo recursivo

Devolve o comprimento de uma sscó máxima de $X[1..i]$ e $Y[1..j]$.

REC-LCS-LENGTH (X, i, Y, j)

```
1 se  $i = 0$  ou  $j = 0$  então devolva 0
2 se  $X[i] = Y[j]$ 
3   então  $c[i, j] \leftarrow \text{REC-LCS-LENGTH}(X, i-1, Y, j-1) + 1$ 
4   senão  $q_1 \leftarrow \text{REC-LCS-LENGTH}(X, i-1, Y, j)$ 
5        $q_2 \leftarrow \text{REC-LCS-LENGTH}(X, i, Y, j-1)$ 
6       se  $q_1 \geq q_2$ 
7         então  $c[i, j] \leftarrow q_1$ 
8         senão  $c[i, j] \leftarrow q_2$ 
9 devolva  $c[i, j]$ 
```

Consumo de tempo

$T(m, n) :=$ número de comparações feitas por REC-LCS-LENGTH (X, m, Y, n) no pior caso

Recorrência

$$T(0, n) = 0$$

$$T(m, 0) = 0$$

$$T(m, n) \geq T(m-1, n) + T(m, n-1) + 1 \text{ para } n \geq 1 \text{ e } m \geq 1$$

A que classe Ω pertence $T(m, n)$?

Recorrência

Note que $T(m, n) = T(n, m)$ para $n = 0, 1, \dots$ e $m = 0, 1, \dots$

Seja $k := \min\{m, n\}$. Temos que

$$T(m, n) \geq T(k, k) \geq S(k),$$

onde

$$S(0) = 0$$

$$S(k) = 2S(k-1) + 1 \text{ para } k = 1, 2, \dots$$

$$S(k) \text{ é } \Theta(2^k) \Rightarrow T(m, n) \text{ é } \Omega(2^{\min\{m, n\}})$$

$T(m, n)$ é **exponencial**

Conclusão

O consumo de tempo do algoritmo REC-LEC-LENGTH é $\Omega(2^{\min\{m,n\}})$.

Programação dinâmica

Cada subproblema, comprimento de uma sscó máxima de

$$X[1..i] \text{ e } Y[1..j],$$

é resolvido **uma só** vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[4,6]$ preciso de ...

$c[4,5]$, $c[3,6]$ e de $c[3,5]$.

Calcule todos os $c[i,j]$ com $i = 1$ e $j = 0, 1, \dots, n$,
 depois todos com $i = 2$ e $j = 0, 1, \dots, n$,
 depois todos com $i = 3$ e $j = 0, 1, \dots, n$,
 etc.

Programação dinâmica

	1	2	3	4	5	6	7	8	j
1	0	0	0	0	0	0	0	0	
2	0								
3	0				★	★			
4	0				★	??			
5	0								
6	0								
7	0								
8	0								
i									

Simulação

	Y	B	D	C	A	B	A	j
X	0	1	2	3	4	5	6	
0	0	0	0	0	0	0	0	
A 1	0	??						
B 2	0							
C 3	0							
B 4	0							
D 5	0							
A 6	0							
B 7	0							
X	Y	B	D	C	A	B	A	j
0	0	1	2	3	4	5	6	
0	0	0	0	0	0	0	0	

Algoritmo de programação dinâmica

Devolve o comprimento de uma ssc máxima de $X[1..m]$ e $Y[1..n]$.

LEC-LENGTH (X, m, Y, n)

```

1  para  $i \leftarrow 0$  até  $m$  faça  $c[i,0] \leftarrow 0$ 
2  para  $j \leftarrow 1$  até  $n$  faça  $c[0,j] \leftarrow 0$ 
3  para  $i \leftarrow 1$  até  $m$  faça
4    para  $j \leftarrow 1$  até  $n$  faça
5      se  $X[i] = Y[j]$ 
6        então  $c[i,j] \leftarrow c[i-1,j-1] + 1$ 
7        senão se  $c[i-1,j] \geq c[i,j-1]$ 
8          então  $c[i,j] \leftarrow c[i-1,j]$ 
9          senão  $c[i,j] \leftarrow c[i,j-1]$ 
10 devolva  $c[m,n]$ 
    
```

Consumo de tempo: $O(mn)$

Conclusão

O consumo de tempo do algoritmo LEC-LENGTH é $\Theta(mn)$.

Subsequência comum máxima

	Y	B	D	C	A	B	A	
X	0	1	2	3	4	5	6	j
0	★	★	★	★	★	★	★	
A 1	★	←	←	←	↖	↑	↖	
B 2	★	↖	↑	↑	←	↖	↑	
C 3	★	←	←	↖	↑	←	←	
B 4	★	↖	←	←	←	↖	↑	
D 5	★	←	↖	←	←	←	←	
A 6	★	←	←	←	↖	←	↖	
B 7	★	↖	←	←	←	↖	←	

Algoritmo de programação dinâmica

LEC-LENGTH (X, m, Y, n)

```

1  para  $i \leftarrow 0$  até  $m$  faça  $c[i,0] \leftarrow 0$ 
2  para  $j \leftarrow 1$  até  $n$  faça  $c[0,j] \leftarrow 0$ 
3  para  $i \leftarrow 1$  até  $m$  faça
4    para  $j \leftarrow 1$  até  $n$  faça
5      se  $X[i] = Y[j]$ 
6        então  $c[i,j] \leftarrow c[i-1,j-1] + 1$ 
7         $b[i,j] \leftarrow "$ ↖ $"$ 
8        senão se  $c[i-1,j] \geq c[i,j-1]$ 
9          então  $c[i,j] \leftarrow c[i-1,j]$ 
10          $b[i,j] \leftarrow "$ ← $"$ 
11        senão  $c[i,j] \leftarrow c[i,j-1]$ 
12          $b[i,j] \leftarrow "$ ← $"$ 
13 devolva  $c$  e  $b$ 
    
```

Consumo de tempo: $O(mn)$

GET-LCS ($X, m, n, b, \text{máxcomp}$)

```

1   $k \leftarrow \text{máxcomp}$ 
2   $i \leftarrow m$ 
3   $j \leftarrow n$ 
4  enquanto  $i > 0$  e  $j > 0$  faça
5      se  $b[i, j] = \nwarrow$ 
6          então  $Z[k] \leftarrow X[i]$ 
7               $k \leftarrow k - 1$     $i \leftarrow i - 1$     $j \leftarrow j - 1$ 
8      senão se  $b[i, j] = \leftarrow$ 
9          então  $i \leftarrow i - 1$ 
10         senão  $j \leftarrow j - 1$ 
11 devolva  $Z$ 
    
```

Consumo de tempo é $O(m + n)$ e $\Omega(\min\{m, n\})$.

Exercício 20.A

Escreva um algoritmo para decidir se $\langle z_1, \dots, z_k \rangle$ é subsequência de $\langle x_1, \dots, x_m \rangle$. Prove rigorosamente que o seu algoritmo está correto.

Exercício 20.B

Suponha que os elementos de uma sequência $\langle a_1, \dots, a_n \rangle$ são distintos dois a dois. Quantas subsequências tem a sequência?

Exercício 20.C

Uma subsequência crescente Z de uma sequência X e é *máxima* se não existe outra subsequência crescente mais longa. A subsequência $\langle 5, 6, 9 \rangle$ de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$ é máxima? Dê uma sequência crescente máxima de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$. Mostre que o algoritmo "guloso" óbvio não é capaz, em geral, de encontrar uma subsequência crescente máxima de uma sequência dada. (Algoritmo guloso óbvio: escolha o menor elemento de X ; a partir daí, escolha sempre o próximo elemento de X que seja maior ou igual ao último escolhido.)

Exercício 20.D

Escreva um algoritmo de programação dinâmica para resolver o problema da subsequência crescente máxima.

Mais exercícios

Exercício 20.E [CLRS 15.4-5]

Mostre como o algoritmo da subsequência comum máxima pode ser usado para resolver o problema da subsequência crescente máxima de uma sequência numérica. Dê uma delimitação justa, em notação Θ , do consumo de tempo de sua solução.

Exercício 20.F [Printing neatly, CLRS 15-2]

Considere a sequência P_1, P_2, \dots, P_n de palavras que constitui um parágrafo de texto. A palavra P_i tem l_i caracteres. Queremos imprimir as palavras em linhas, na ordem dada, de modo que cada linha tenha no máximo M caracteres. Se uma determinada linha contém as palavras P_i, P_{i+1}, \dots, P_j (com $i \leq j$) e há exatamente um espaço entre cada par de palavras consecutivas, o número de espaços no fim da linha é

$$M - (l_i + 1 + l_{i+1} + 1 + \dots + 1 + l_j).$$

É claro que não devemos permitir que esse número seja negativo. Queremos minimizar, com relação a todas as linhas exceto a última, a soma dos cubos dos números de espaços no fim de cada linha. (Assim, se temos linhas $1, 2, \dots, L$ e b_p espaços no fim da linha p , queremos minimizar $b_1^3 + b_2^3 + \dots + b_{L-1}^3$.) Dê um exemplo para mostrar que algoritmos inocentes não resolvem o problema. Dê um algoritmo de programação dinâmica que resolva o problema. Qual a "optimal substructure property" para esse problema? Faça uma análise do consumo de tempo do algoritmo.

Mais programação dinâmica

KT 6.4

Aproveite para olhar todo o Cap 6 do KT, que é sobre programação dinâmica.

= "recursão-com-tabela"

= transformação inteligente de recursão em iteração

Mochila

Dados dois vetores $x[1..n]$ e $w[1..n]$, denotamos por $x \cdot w$ o **produto escalar**

$$w[1]x[1] + w[2]x[2] + \dots + w[n]x[n].$$

Suponha dado um número inteiro não-negativo W e vetores positivos $w[1..n]$ e $v[1..n]$.

Uma **mochila** é qualquer vetor $x[1..n]$ tal que

$$x \cdot w \leq W \quad \text{e} \quad 0 \leq x[i] \leq 1 \quad \text{para todo } i$$

O **valor** de uma mochila é o número $x \cdot v$.

Uma mochila é **ótima** se tem valor máximo.

Problema booleano da mochila

Uma mochila $x[1..n]$ tal que $x[i] = 0$ ou $x[i] = 1$ para todo i é dita **booleana**.

Problema (Knapsack Problem): Dados (w, v, n, W) , encontrar uma **mochila booleana ótima**.

Exemplo: $W = 50, n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor = 840
x	1	0	0	1	valor = 940
x	0	1	1	0	valor = 1000

Subestrutura ótima

Suponha que $x[1..n]$ é **mochila booleana ótima** para o problema (w, v, n, W) .

Se $x[n] = 1$

então $x[1..n-1]$ é **mochila booleana ótima** para $(w, v, n-1, W - w[n])$

senão $x[1..n-1]$ é **mochila booleana ótima** para $(w, v, n-1, W)$

NOTA. Não há nada de especial acerca do índice n .
Uma afirmação semelhante vale para qualquer índice i .

Simplificação

Problema:

encontrar o **valor** de uma mochila booleana ótima.

$t[i, Y]$ = valor de uma mochila booleana ótima para (w, v, i, Y)
= valor da expressão $x \cdot v$ sujeito às restrições

$$x \cdot w \leq Y,$$

onde x é uma mochila booleana ótima

Possíveis valores de Y : $0, 1, 2, \dots, W$

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

$t[i, Y] = t[i-1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\}$ se $w[i] \leq Y$

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W) .

REC-MOCHILA (w, v, n, W)

1 se $n = 0$ ou $W = 0$

2 então devolva 0

3 se $w[n] > W$

4 então devolva REC-MOCHILA $(w, v, n-1, W)$

5 $a \leftarrow$ REC-MOCHILA $(w, v, n-1, W)$

6 $b \leftarrow$ REC-MOCHILA $(w, v, n-1, W-w[n]) + v[n]$

7 devolva $\max\{a, b\}$

Consumo de tempo no pior caso é $\Omega(2^n)$

Por que demora tanto?

O mesmo subproblema é resolvido muitas vezes.

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido **uma só** vez.

Em que ordem calcular os componentes da tabela t ?

Olhe a recorrência e pense...

$t[i, Y] = t[i-1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\}$ se $w[i] \leq Y$

Programação dinâmica

	0	1	2	3	4	5	6	7	Y
0	0	0	0	0	0	0	0	0	
1	0								
2	0	★	★	★	★	★			
3	0					??			
4	0								
5	0								
6	0								
7	0								

i

Exemplo

$$W = 5 \text{ e } n = 4$$

		1	2	3	4	
w		4	2	1	3	
v		500	400	300	450	
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

i

Exemplo

$$W = 5 \text{ e } n = 4$$

		1	2	3	4	
w		4	2	1	3	
v		500	400	300	450	
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4	0					

i

		0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	0	0	0				
2	0						
3	0						
4	0						

Exemplo

$$W = 5 \text{ e } n = 4$$

		1	2	3	4	
w		4	2	1	3	
v		500	400	300	450	
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0				
3	0					
4	0					

i

		0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	0	0	0	0	500	500	
2	0	0	400				
3	0						
4	0						

i

Exemplo

$$W = 5 \text{ e } n = 4$$

		1	2	3	4	
w		4	2	1	3	
v		500	400	300	450	
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	700	700	800
4	0	300	400	700	700	850

i

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W) .

MOCHILA-BOOLEANA (w, v, n, W)

```
1 para  $Y \leftarrow 0$  até  $W$  faça
2    $t[0, Y] \leftarrow 0$ 
3   para  $i \leftarrow 1$  até  $n$  faça
4      $a \leftarrow t[i-1, Y]$ 
5     se  $w[i] > Y$ 
6       então  $b \leftarrow 0$ 
7       senão  $b \leftarrow t[i-1, Y-w[i]] + v[i]$ 
8      $t[i, Y] \leftarrow \max\{a, b\}$ 
9 devolva  $t[n, W]$ 
```

Consumo de tempo é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:

O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o “tamanho” de W é $\lg W$ e não W (tente multiplicar $w[1], \dots, w[n]$ e W por 1000)

Se W é $\Omega(2^n)$ o consumo de tempo é $\Omega(n2^n)$, mais lento que o algoritmo força bruta!

Obtenção da mochila

MOCHILA (w, n, W, t)

```
1  $Y \leftarrow W$ 
2 para  $i \leftarrow n$  decrescendo até 1 faça
3   se  $t[i, Y] = t[i-1, Y]$ 
4     então  $x[i] \leftarrow 0$ 
5     senão  $x[i] \leftarrow 1$ 
6      $Y \leftarrow Y - w[i]$ 
7 devolva  $x$ 
```

Consumo de tempo é $\Theta(n)$.

Versão recursiva

MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)

```
1 para  $i \leftarrow 0$  até  $n$  faça
2   para  $Y \leftarrow 0$  até  $W$  faça
3      $t[i, Y] \leftarrow \infty$ 
3 devolva LOOKUP-MOC  $(w, v, n, W)$ 
```

Versão recursiva

```
LOOKUP-MOC ( $w, v, i, Y$ )
1  se  $t[i, Y] < \infty$ 
2    então devolva  $t[i, Y]$ 
3  se  $i = 0$  ou  $Y = 0$  então  $t[i, Y] \leftarrow 0$ 
   senão
4    se  $w[i] > Y$ 
       então
5       $t[i, Y] \leftarrow \text{LOOKUP-MOC}(w, v, i-1, Y)$ 
       senão
6       $a \leftarrow \text{LOOKUP-MOC}(w, v, i-1, Y)$ 
7       $b \leftarrow \text{LOOKUP-MOC}(w, v, i-1, Y-w[i]) + v[i]$ 
8       $t[i, Y] \leftarrow \max\{a, b\}$ 
9  devolva  $t[i, Y]$ 
```

Exercício das bandeiras

No dia da Bandeira na Rússia o proprietário de uma loja decidiu decorar a vitrine de sua loja com faixas de tecido das cores branca, azul e vermelha.

Ele deseja satisfazer as seguintes condições: faixas da mesma cor não podem ser colocadas uma ao lado da outra. Uma faixa azul sempre está entre uma branca e uma vermelha, ou uma vermelha e uma branca.

Escreva um programa que, dado o número n de faixas a serem colocadas na vitrine, calcule o número de maneiras de satisfazer as condições do proprietário.

Exemplo: Para $n = 3$, o resultado são as seguintes combinações: BVB, VBV, BAV, VAB.